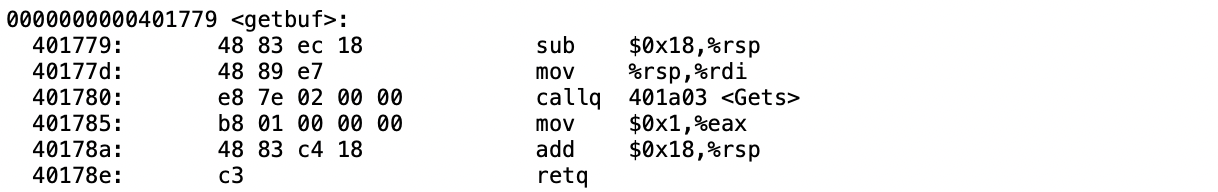
**시스템소프트웨어(CB2400108-059) - HW3:Attacklab**

정보컴퓨터공학부 202255513 김대욱

**[ phase\_1 ]**

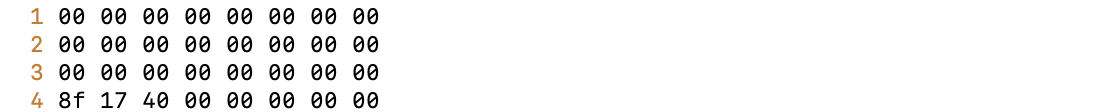


getbuf() 함수에서, 0x18 byte만큼 rsp를 뺀 후, Gets() 함수를 호출하는 과정이 있다. Gets() 함수에서 buffer에 0x18 byte 이상의 값을 넣게 된다면, return address를 침범하여 stack overflow가 발생할 수 있다.

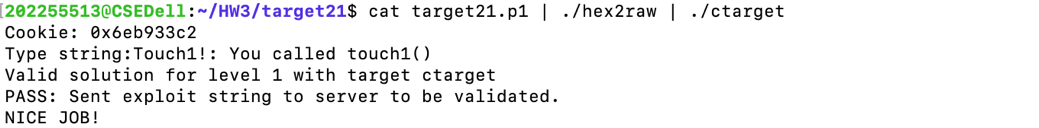
A picture containing text, receipt, font, algebra

Description automatically generated

getbuf() 함수가 실행되었을 때, stack에 있는 return address를 touch1() 함수의 시작 주소로 덮어쓰면 정상적으로 실행된다. touch1() 의 시작 주소는 0x40178f 이고, little endian 형식으로 **target21.p1** 파일을 생성한다. 0x18 만큼의 buffer를 비워야하므로, 24 byte를 비워두고 주소를 적으면 아래와 같다.



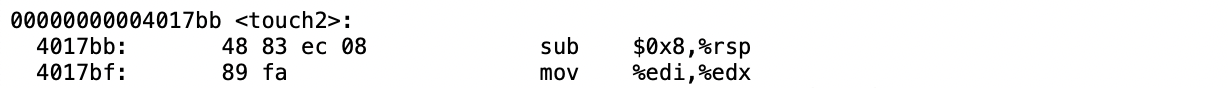
**cat target21.p1 | ./hex2raw | ./ctarget** 명령어를 통해 정답여부를 확인하면 아래와 같다.

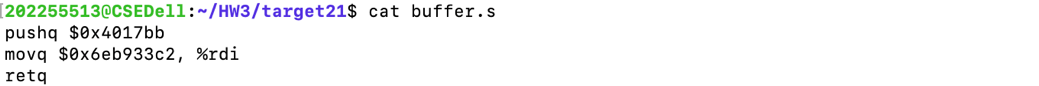


**[ phase\_2 ]**

touch2 () 함수를 call 한 이후, %edi register의 값을 cookie.txt의 값으로 바꾼다.







cookie 값과 touch2() 함수의 시작주소 값을 이용하여 buffer.s 어셈블리 파일을 만든 후, 1) **gcc -c buffer.s**

2) **objdump -d buffer.o > buffer.txt** 명령어를 사용하면, 아래와 같은 값을 얻을 수 있다.

A picture containing text, screenshot, font, algebra

Description automatically generated

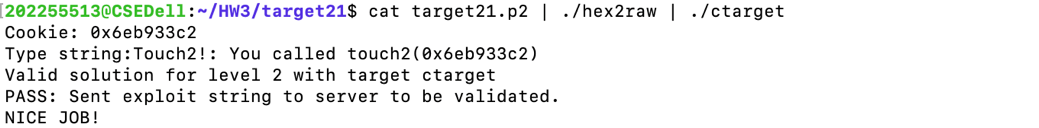
A screenshot of a computer code

Description automatically generated with low confidence

rdi 값을 구해주면, 0x55641138 이다. Little Endian임을 주의하여 buffer에 들어갈 정답 파일을 만들면 아래와 같다.



**cat target21.p2 | ./hex2raw | ./ctarget** 명령어를 통해 정답여부를 확인하면 아래와 같다.

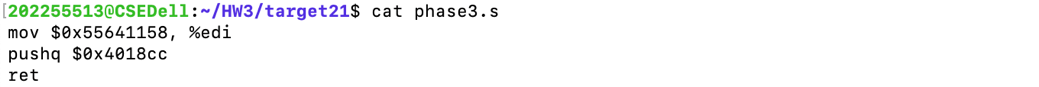


**[ phase\_3 ]**

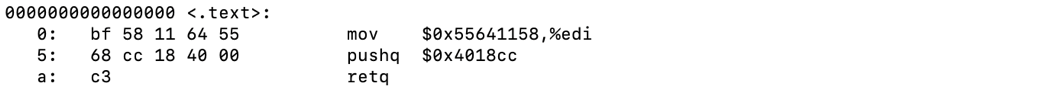
함수 실행과 함께 문자열의 주소를 인자로 전달해야 한다. 먼저 **gdb ctarget** 명령어로 gdb를 실행시킨 후**, break getbuf** 명령어로 breakpoint를 건 뒤, **info register**를 통해 $rsp의 주소값을 가져온다.

A screenshot of a computer code

Description automatically generated with medium confidence



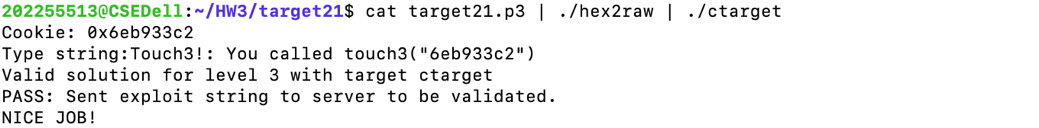
해당 $rsp의 주소값인 0x55641150 에 0x8을 더한 값인 0x55641158 touch3() 함수의 시작주소 값을 이용하여 phase3.s 어셈블리 파일을 만든 후, 1) **gcc -c phase3.s** 2) **objdump -d phase3.o > phase3.txt** 명령어를 사용하면, 아래와 같은 값을 얻을 수 있다.



buffer의 크기인 0x18 만큼을 만든 후, phase2처럼 4번째 줄에는 rdi 값을 넣어주고, 5번째 줄에는 cookie의 값을 ASCII 코드로 변환하여 넣어주면 된다.



**cat target21.p3 | ./hex2raw | ./ctarget** 명령어를 통해 정답여부를 확인하면 아래와 같다.



**[ phase\_4 ]**

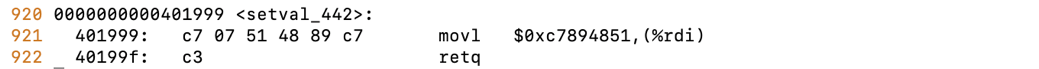
phase\_4는 앞의 세 문제와 다르게 ctarget이 아닌, rtarget을 이용한다. popq %rax, ret를 통해 스택에 저장된 cookie값을 %rax에 저장하고, movq %rax, %rdi를 통해 값을 옮겨준 뒤 touch2() 함수를 불러오면 된다.

phase\_4 문제는 0x18 만큼의 값을 비워두고, 차례로 명령을 넣어주면 된다. popq %rax 명령(58)의 위치는 0x401984 + 4 이므로, little endian으로 한줄을 채운다.

A picture containing text, screenshot, receipt, font

Description automatically generated

그 뒷줄은 cookie 값을 넣어주고, %rdi -> %rdi 작업을 해야한다. 따라서 48 89 c7 가젯으로 주소값을 찾아보면, 0x401979 + 3 가 된다. 마지막 줄은 touch2() 함수의 시작 주소값을 넣어주면 된다.

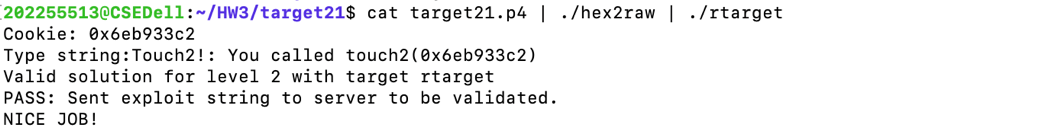


아래는 정답 코드이다.

A picture containing text, screenshot, design, algebra

Description automatically generated

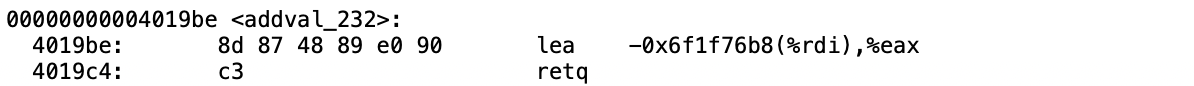
**cat target21.p4 | ./hex2raw | ./rtarget** 명령어를 통해 정답여부를 확인하면 아래와 같다.



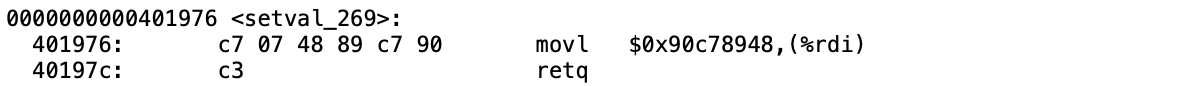
**[ phase\_5 ]**

phase\_4는 %eax의 값을 %esi로 이동시키고, %rsp 값을 %edi로 이동시킨 후 add\_xy 함수를 이용하는 문제이다. rtarget을 disas 하여 명령어를 잘 살펴보면 된다. 유효한 가젯은 90, c3 과 pdf에 있는 nop instruction을 활용하여 유효한 가젯을 살펴본다.

**%rsp -> %rax** 의 기계어 명령은 **"48 89 e0"** 이고, 주소 값은 **0x4019c0** 이다.

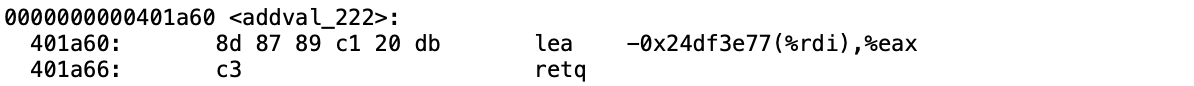


**%rax -> %rdi** 의 기계어 명령은 **"48 89 c7"** 이고, 주소 값은 **0x401978** 이다.

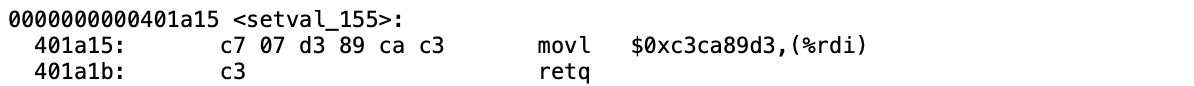


**pop %rax** 명령어는 phase\_4에서 참고하면, 0x401988이고, offset은 48이다.

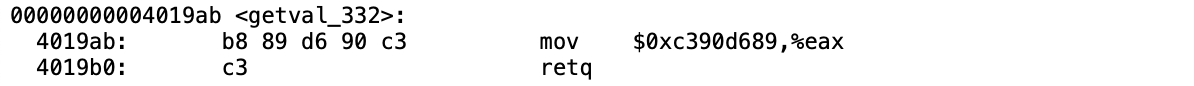
**%eax -> %ecx** 의 기계어 명령은 **"89 c1"** 이고, 주소 값은 **0x401a62** 이다.

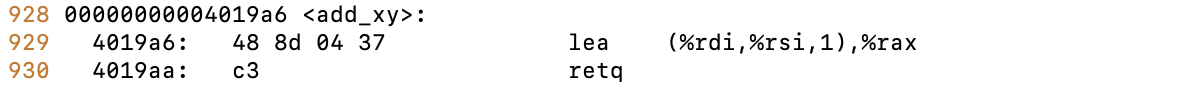


**%ecx -> %edx** 의 기계어 명령은 **"89 ca"** 이고, 주소 값은 **0x401a18** 이다.



**%edx -> %esi** 의 기계어 명령은 **"89 d6"** 이고, 주소 값은 **0x4019ac** 이다.





touch3() 함수의 주소 값과, cookie 값의 ASCII 변환 값은 위에서 참고하면 아래와 같은 정답 코드가 완성된다.

A picture containing screenshot, text, design

Description automatically generated

**cat target21.p5 | ./hex2raw | ./rtarget** 명령어를 통해 정답여부를 확인하면 아래와 같다.

A black text on a white background

Description automatically generated with low confidence